Estudos sobre Redes Neurais Convolucionais

Disciplina PSI5886

Prof. Emilio Del Moral Hernandez

Bruno Canale
Bruno Giordano
Fábio Sancinetti
Wanderson Ferreira

December 6, 2016

Objetivos do Trabalho

Os objetivos principais do trabalho foram:

- Entender como as **MLPs** clássicas evoluíram para o que hoje é conhecido como Deep Learning
- Estudar a estrutura e o funcionamento de Redes Neurais Convolucionais 2D
- Entender como a informação é transformada dentro da rede neural ao avançar nas camadas mais profundas
- Extrapolar esse conhecimento para redes mais clássicas como a **MLP** estudada durante a disciplina.

Introdução e Motivação

- Cybenko prova que uma rede neural MLP com uma camada escondida e com número arbitrário de neurônios consegue aproximar qualquer função
- Considerando tal resultado, por que tentar fazer redes neurais profundas?
- Delalleau e Bengio fizeram um estudo teórico [1] com neurônios simples (chamados de sum-product units) comparando a quantidade de neurônios necessária para uma aproximação com redes "superficiais" e "profundas"

[1] O. Delalleau, Y. Bengio - *Shallow vs Deep Sum-Product Networks*, Neural Information Processing Systems

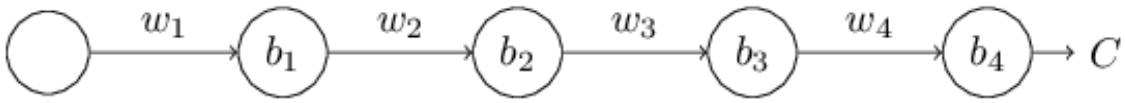
Introdução e Motivação

- De maneira simplificada, o trabalho [2] conclui que, para uma classe de funções:
  It shows that some deep sumproduct network with n inputs and depth $O(\log n)$ can represent with $O(n)$ units what would require $O(2^{\sqrt{n}})$ units for a depth-2 network.
- Em outras palavras, conclui-se que para alguns tipos de funções a utilização de redes neurais como postuladas por Cybenko requer um número de neurônios consideravelmente maior à uma rede neural com mais camadas escondidas
- Tarefas clássicas de inteligência artificial (como reconhecimento de imagens) se encaixam em algumas dessas premissas, justificando a utilização de arquiteturas profundas nesses problemas
- O que acontece se aumentarmos arbritariamente o número de camadas escondidas em uma MLP clássica como visto em aula?

[2] O. Delalleau, Y. Bengio - *Shallow vs Deep Sum-Product Networks*, Neural Information Processing Systems

Análise preliminar da MLP clássica vista em sala

Para entender os problemas no aprendizado de redes com grande número de camadas escondidas em uma MLP clássica, vamos relembrar alguns conceitos de rede neurais com a arquitetura profunda mais simples possível [3] :



Seja a_i o output do i-ésimo neurônio do modelo. Ou seja:

$$a = \begin{cases} a_1 = \sigma\left(w_1 x + b_1\right) = \sigma(z_1) \\ a_i = \sigma\left(w_i a_{i-1} + b_i\right) = \sigma(z_i) \quad i > 1 \end{cases}$$

Onde $\sigma(.)$ é a função de ativação do i-ésimo neurônio. A derivada parcial de C em relação à w_1 pode ser calculada como:

$$\frac{\partial C}{\partial w_1} = x \cdot \sigma^{'}(z_1) \cdot w_2 \cdot \sigma^{'}(z_2) \cdot w_3 \cdot \sigma^{'}(z_3) \cdot w_4 \cdot \sigma^{'}(z_4) \cdot \frac{\partial C}{\partial a_4}$$

[3] Michael A. Nielsen. - *Neural Networks and Deep Learning*, Determination Press, 2015

Análise preliminar da MLP clássica vista em sala

Há dois problemas cruciais consequentes da expressão da derivada parcial de C em relação à w_1.

- Função de perda C:
  $C = \frac{(y-a_4)^2}{2}$ como erro quadrático médio não é muito adequado para problemas de classificação:

$$\frac{\partial C}{\partial w_4} = a_3(a_4 - y_{ref})\sigma'(z_4)$$

  Para $a_4 \to \{1,0\}$, $\sigma'(z_4) \to 0$. Assim, se $a_4 \to \{1,0\}$, $\frac{\partial C}{\partial w_4} \to 0$. Se $a_4 \to y_{ref}$, é interessante que os pesos se modifiquem menos pois a_4 já está correto. Entretanto, se a_4 estiver próximo do valor binário contrário à y_{ref}, $\sigma'(z_4)$ impede uma correção grande nos pesos para corrigir a_4.
  Alternativa: Cross-Entropy Loss

Análise preliminar da MLP clássica vista em sala

Cross-entropy loss:

$$C = -\left(y_{ref} \ln a_4 + (1 - y_{ref}) \ln (1 - a_4) \right)$$

Como $y_{ref} = \{1, 0\}$, podemos perceber que $C \to 0$ se $a_4 = y_{ref}$ e $C \to \infty$ se a_4 se aproxima do valor binário contrário de y_{ref}. A propriedade mais interessante dessa função de perda está em sua derivada parcial:

$$\frac{\partial C}{\partial w_4} = a_3(a_4 - y_{ref})$$

Como podemos ver, a derivada independe de $\sigma'(z_4)$. Assim, não há mais o problema mencionado anteriormente. Nas implementações práticas percebe-se que tal função de perda resulta em um treinamento consideravelmente mais rápido para problemas de classificação. Note que esse problema não é especificamente relacionado à redes profundas já que só se manifesta na última camada. Experimentalmente, verificou-se que para arquiteturas profundas diferentes camadas aprendiam em taxas diferentes de acordo com sua profundidade. Vamos voltar a analisar $\frac{\partial C}{\partial w_1}$:

Análise preliminar da MLP clássica vista em sala

Considere $\frac{\partial C}{\partial w_1}$:

$$\frac{\partial C}{\partial w_1} = x \cdot \sigma^{'}(z_1) \cdot w_2 \cdot \sigma^{'}(z_2) \cdot w_3 \cdot \sigma^{'}(z_3) \cdot w_4 \cdot \sigma^{'}(z_4) \cdot \frac{\partial C}{\partial a_4} \rightarrow$$

$$\frac{\partial C}{\partial w_1} = x\left(w_2 w_3 w_4\right) \cdot \left(\sigma^{'}(z_1) \cdot \sigma^{'}(z_2) \cdot \sigma^{'}(z_3) \cdot \sigma^{'}(z_4)\right) \frac{\partial C}{\partial a_4}$$

Sabemos que para $\sigma(z) = \frac{1}{1+e^{z}}$, $\sigma^{'}(z) \leq 0.25$ e que $\sigma^{'}(z) \rightarrow 0$ para certos valores de z. Assim, a multiplicação de vários $\sigma^{'}(z)$ resulta em valores cada vez menores. Quanto maior a "distância" entre a camada e a função de perda C, menor a velocidade de aprendizado. Em particular, se inicializarmos todos os pesos com a mesma distribuição aleatória sem considerar a profundidade da camada em que se encontram, teremos uma inicialização com $\frac{\partial C}{\partial w_1} < \frac{\partial C}{\partial w_2} < \frac{\partial C}{\partial w_3} < \ldots$. Tal problema é uma das principais causas da convergência para mínimos locais em redes profundas, comumente chamado na literatura de *Vanishing Gradient Problem.*

Deep Learning

Durante um bom tempo tal problema impossibilitava a utilização de arquiteturas profundas para machine learning. Em 2006 [4] e 2007 [5] surgiram os primeiros algoritmos bem sucedidos para lidar com esse problema objetivando melhorar a inicialização da rede.
Ambos algoritmos exploravam a ideia da utilização de técnicas de aprendizado não supervisionado para a inicialização dos pesos da rede seguidas de técnicas de aprendizado supervisionado. Tal estratégia talvez não seja mais tão utilizada atualmente em classificadores estado-da-arte, mas certamente foi extremamente importante na época pois estimulou a comunidade científica na pesquisa de novos métodos para treinamento de redes profundas.
Surge, assim, o Deep Learning como conjunto de técnicas para melhorar o treinamento de redes profundas. Indicaremos a seguir algumas técnicas já consolidadas:

---

[4] Hinton et al. - *A fast learning algorithm for deep belief nets*, Neural Computation, 2006

[5] Bengio et al. - *Greedy Layer-Wise Training of Deep Networks*, Neural Information Processing Systems, 2007

Regularização

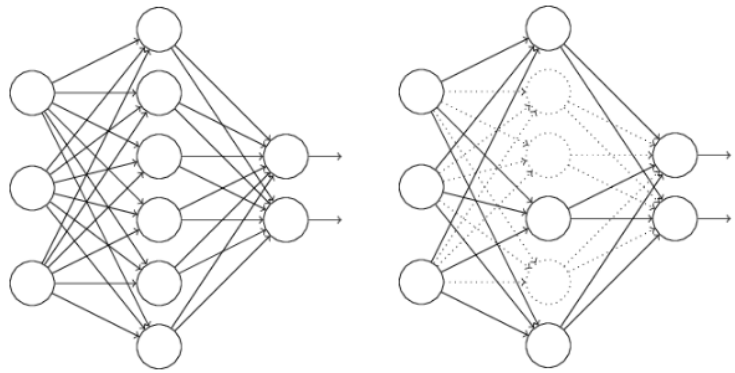
Como vimos, o problema de aprendizado lento ocorre quando $\frac{\partial C}{\partial w_i}$ é pequeno. Considerando $\sigma(z)$ e $\sigma^{'}(z)$, sabemos que isso ocorre criticamente quando o módulo de z é grande. Já que $z = \vec{w}^T \vec{a} + b$ (onde $\vec{a}$ são as ativações da camadas anterior e portanto são menores ou iguais a 1), o módulo de z é grande quando o módulo de $\vec{w}$ é grande.
Assim, para evitar que a rede "caminhe" durante o treinamento para regiões de baixo aprendizado, introduzimos um termo na função de custo C que penaliza pesos grandes. Por exemplo, a regularização L2 utiliza a norma L2 dos pesos $\vec{w}$ como penalidade:

$$C_{reg} = C_{orig} + \lambda \vec{w}^T \vec{w}$$

Onde o parâmetro λ é utilizado como hyper-parâmetro para ponderar a importância desse termo na função de perda. Quanto maior o parâmetro λ, menores serão os pesos encontrados durante o treinamento.

Dropout

Introduzido por Hinton et al. [6], pode ser interpretado simplificadamente como uma técnica para evitar o problema de overfitting. Em cada iteração de treino, há uma probabilidade p para cada neurônio estar ativo (pesos w mantidos) e (1-p) de estar inativo (pesos w zerados).



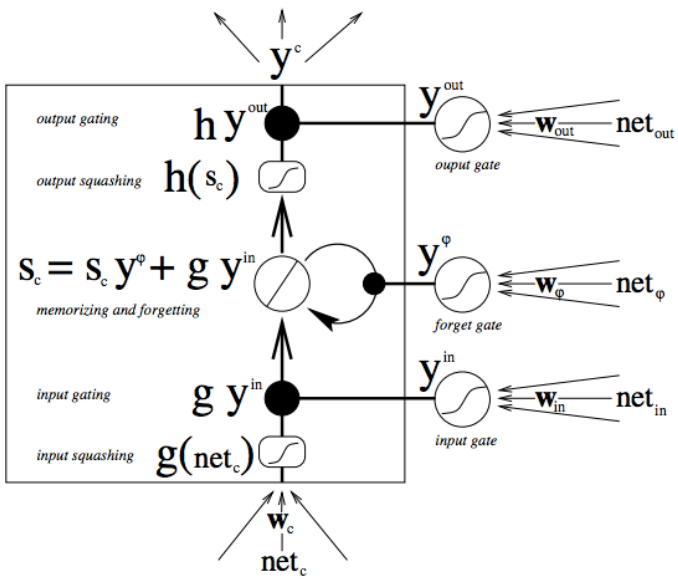
Foi popularizada após a utilização de Krizhevsky et al. [7] em sua solução campeã da competição ImageNet de 2012. ***"This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons."***

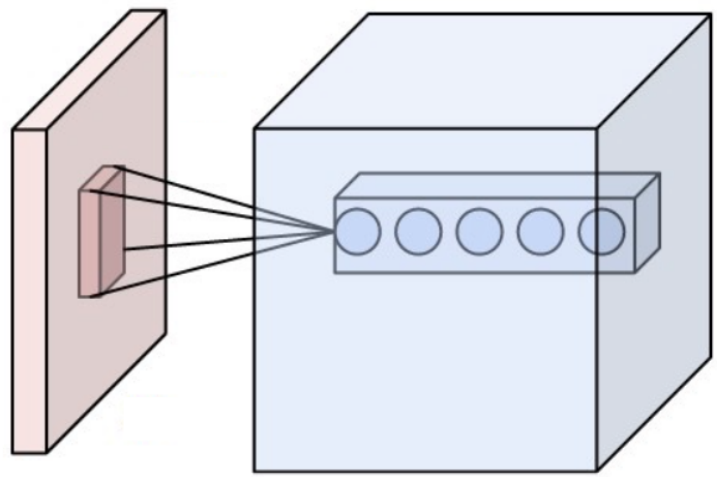
[6] Hinton et al. - *Improving neural networks by preventing co-adaptation of feature detectors*, 2012

[7] Krizhevsky et al. - *ImageNet Classification with Deep Convolutional Neural Networks* 2012

Generalização de arquiteturas de ativação: building blocks

Com o passar dos anos, a comunidade científica foi testando experimentalmente novas metodologias orientadas à resultados sem tanto embasamento teórico como a MLP de Cybenko. Nesse contexto, novas funções de ativação e novas arquiteturas de conexões entre neurônios foram surgindo na tentativa de inserir conhecimento específico de domínio na generalidade das redes neurais. Introduz-se então o conceito de "building blocks", que são pequenos blocos neurais pré-definidos. Alguns exemplos:





Apresentação do Bruno Canale

Visualizando a Convolução

`http://setosa.io/ev/image-kernels/`

Explicação visual sobre Convoluções com demonstrações em Javascript

Visualizando a Convolução

0	0	0
0	1	0
0	0	0

custom

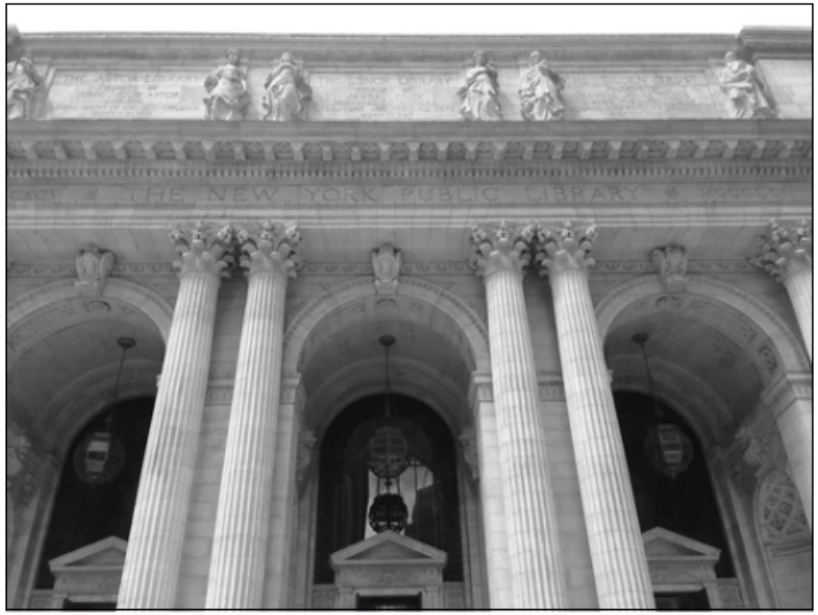
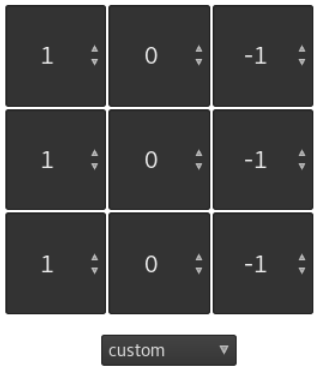


Fig.: Imagem original

Visualizando a Convolução



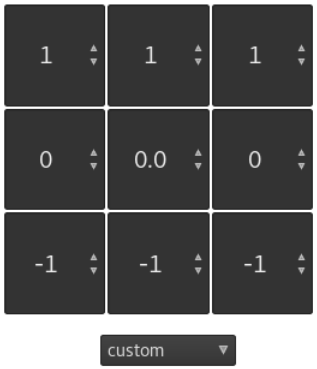
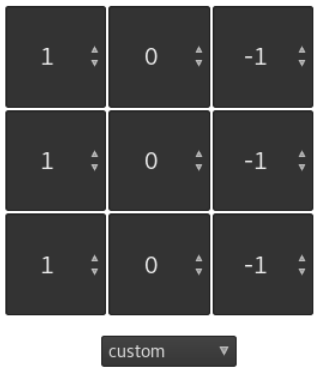


Fig.: Acima: efeito de *Left Sobel*, Abaixo: Efeito *Upper Sobel*

Visualizando a Convolução



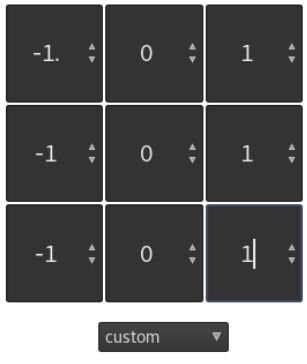
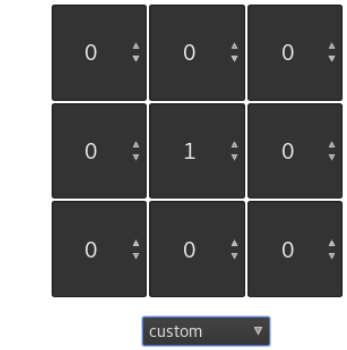
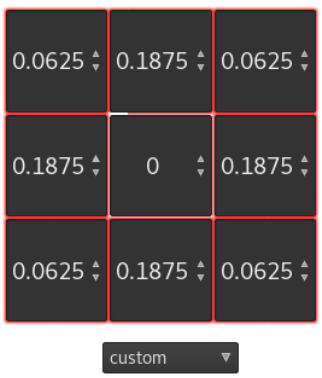


Fig.: Acima: efeito de *Left Sobel*, Abaixo: Efeito *Right Sobel*

Visualizando a Convolução





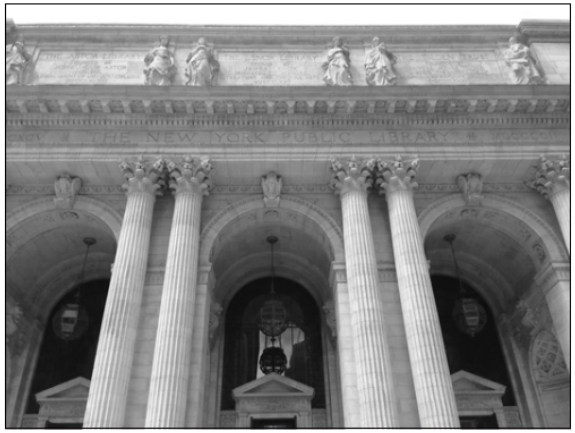


Fig.: Acima: Imagem Original, Abaixo: Efeito *Blur*

Visualizando a Convolução

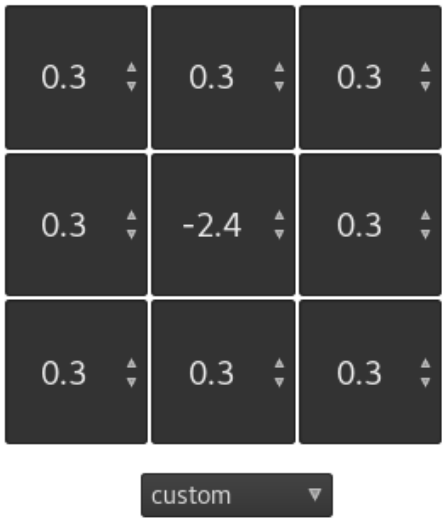
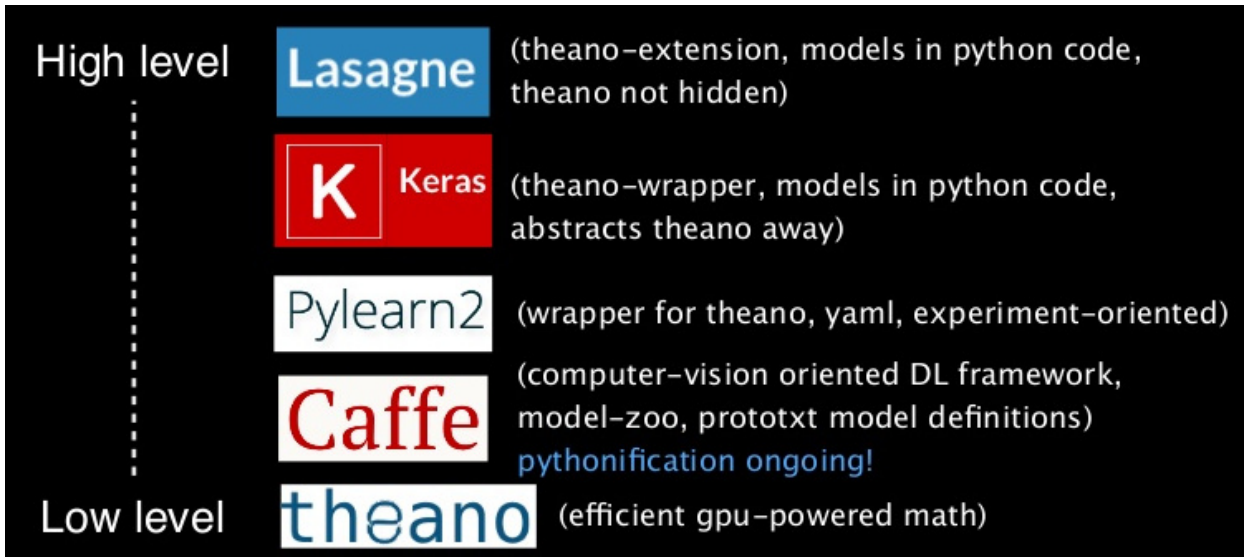


Fig.: Exemplo trivial de detecção de bordas com filtro convolucional

Python - Keras Framework para Machine Learning

* Python - Linguagem de programação gratuita
* Contém uma quantidade muito grande de Frameworks voltados para *Machine Learning*



Keras foi inicialmente desenvolvido como parte de um projeto de pesquisa chamado de ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System)

Keras - Exemplo de implementação MLP

```
modelo = Sequential()
modelo.add(Dense(num-neuronios, init='uniform'))
modelo.add(Activation('tanh'))

modelo.add(Dense(1, init='uniform'))
modelo.add(Activation('linear'))

modelo.compile(learning-rate=0.1, optimizer='sgd]')

modelo.fit(features-treino, target-treino)

modelo.predict(features-teste)
```

Base de dados utilizada - MNIST

A base de dados MNIST é composta por 60.000 exemplos de imagens de digitos em letra cursivas. O dataset é ideal para testes de algoritmos em reconhecimentos de padrões por necessitar pouco pré-processamento.
Mais informações no link: http://yann.lecun.com/exdb/mnist/

```
from keras.datasets import mnist

(X_treino, y_treino), (X_teste, y_teste) = mnist.load_data()
```

Processamento necessário no atributo target

Um processamento padrão é transformar o conjunto de **atributos targets** em um conjunto de variáveis categóricas. O que seriam variáveis categoricas?
Exemplo: Se a lista de targets é composta por: [1.2, 2, 3, 4.2, 4i] e as classes disponiveis são **real, inteiro, imaginario**, então uma matriz de transformação seria:

Table: Conversão para variáveis categoricas

target	real	inteiro	imaginario
1.2	1	0	0
2	0	1	0
3	0	1	0
4.2	1	0	0
4i	0	0	1

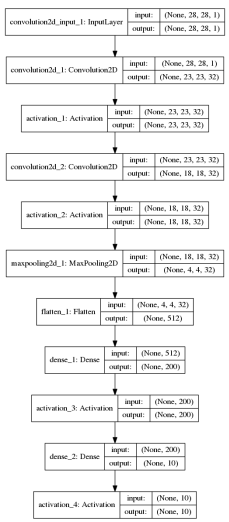
Implementação da Rede Convolucional 2D

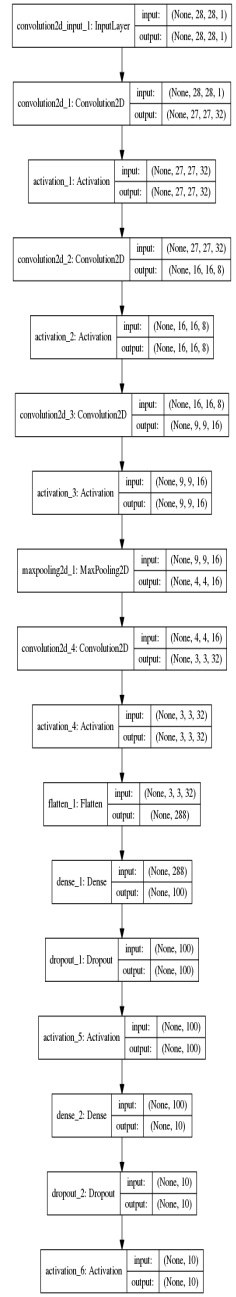
```
from keras.layers import Activation, Dense, Flatten
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D

num_filtros = 32
num_conv = 6
num_pool = 4

modelo = Sequential() # instaciar o modelo

modelo.add(Convolution2D(num_filtros, num_conv, num_conv,
                         border_mode='valid',
                         input_shape=(28, 28, 1)))
modelo.add(Activation('relu'))

# adicao da segunda camada convolucional
modelo.add(Convolution2D(num_filtros, num_conv, num_conv))
modelo.add(MaxPooling2D(pool_size=(num_pool, num_pool)))

# camada que transforma ('comprime') a saida em um array 1D
modelo.add(Flatten())

modelo.add(Dense(100, activation='relu'))
modelo.add(Dense(numero_classes_categoricas),
           activation='softmax')
```

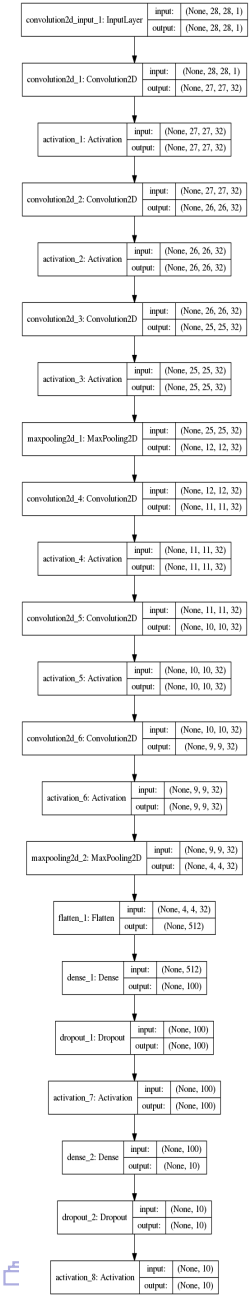
Modelo do Experimento realizado para análise da Rede

- Após implementação a arquitetura foi testada para verificar performance no conjunto de testes.
- Queriamos observar a transformação da imagem de Input após cada camada da ConvNet
- A fim de analisar com mais detalhes, foram adicionadas mais camadas convolucionais no modelo apresentado anteriormente.

Redes e Resultados

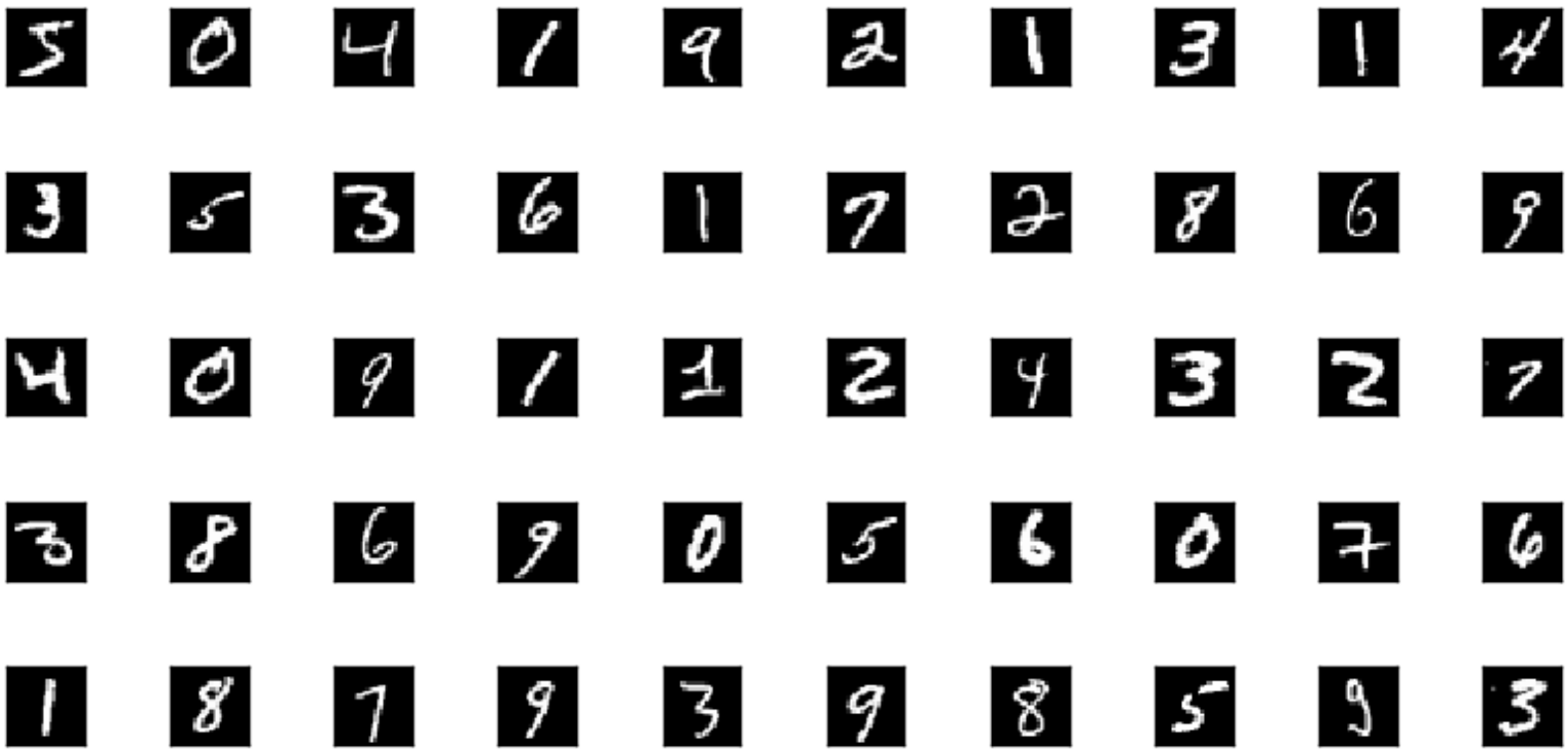




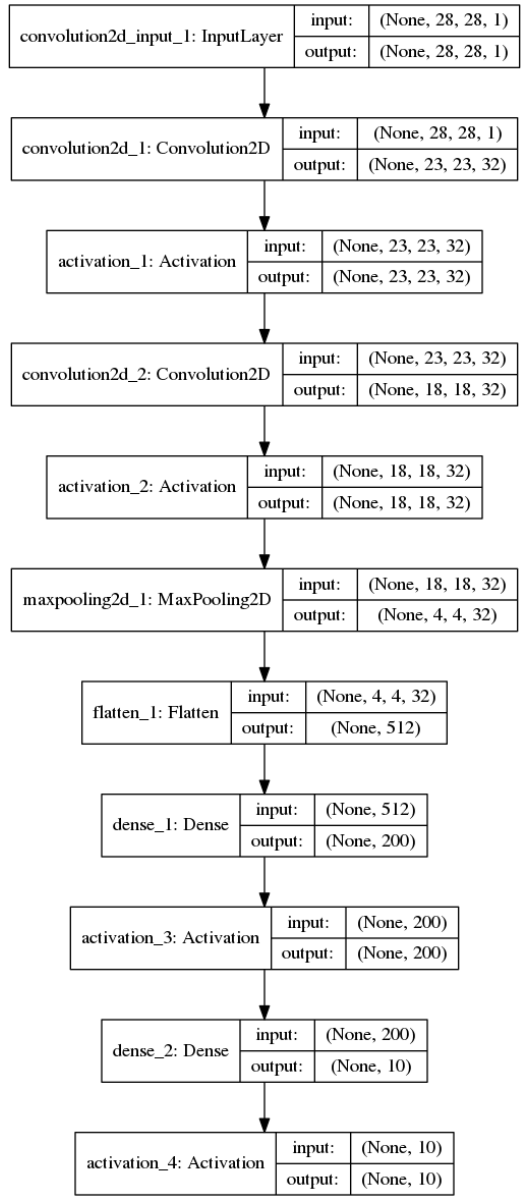


Camada de entrada

MNIST DATASET



Aplicação da CNN



Treinamento

Treinamento

- Épocas = 10
- Itens = 60000
- Tempo = 30 40 minutos

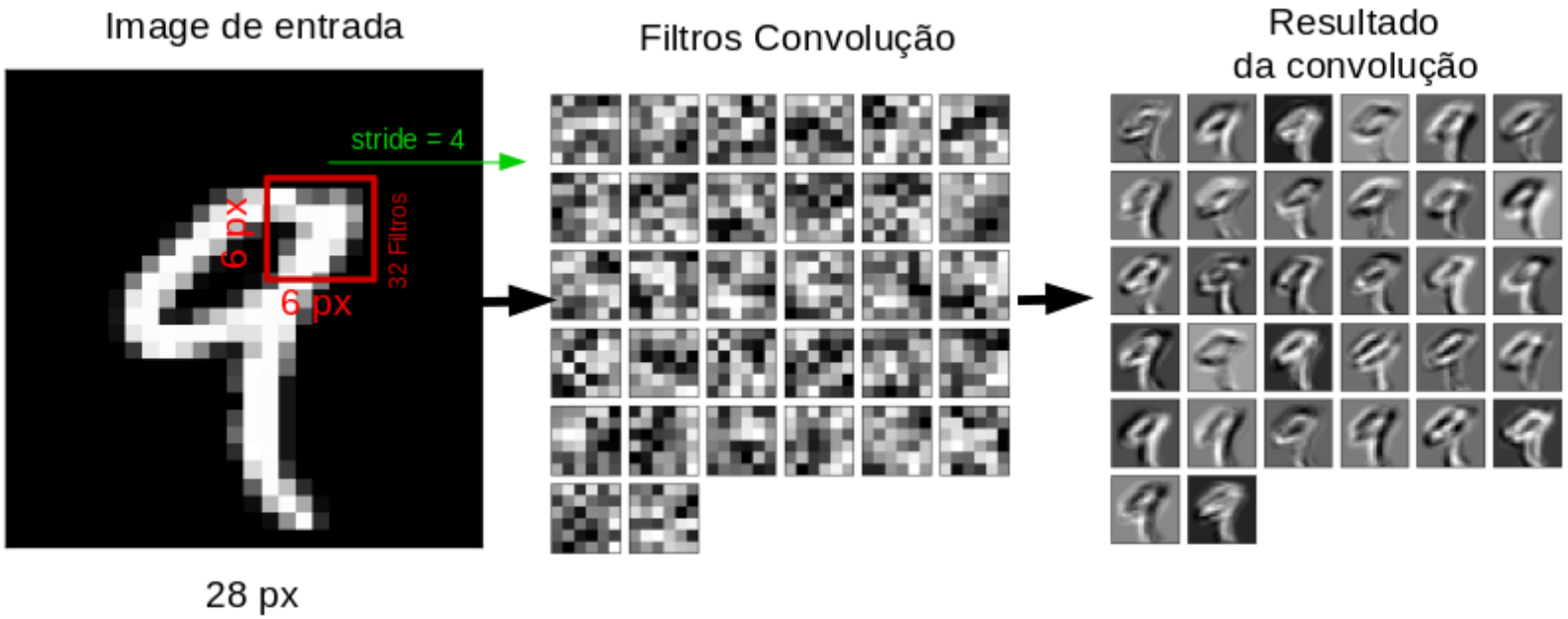
Teste

- Itens = 10000

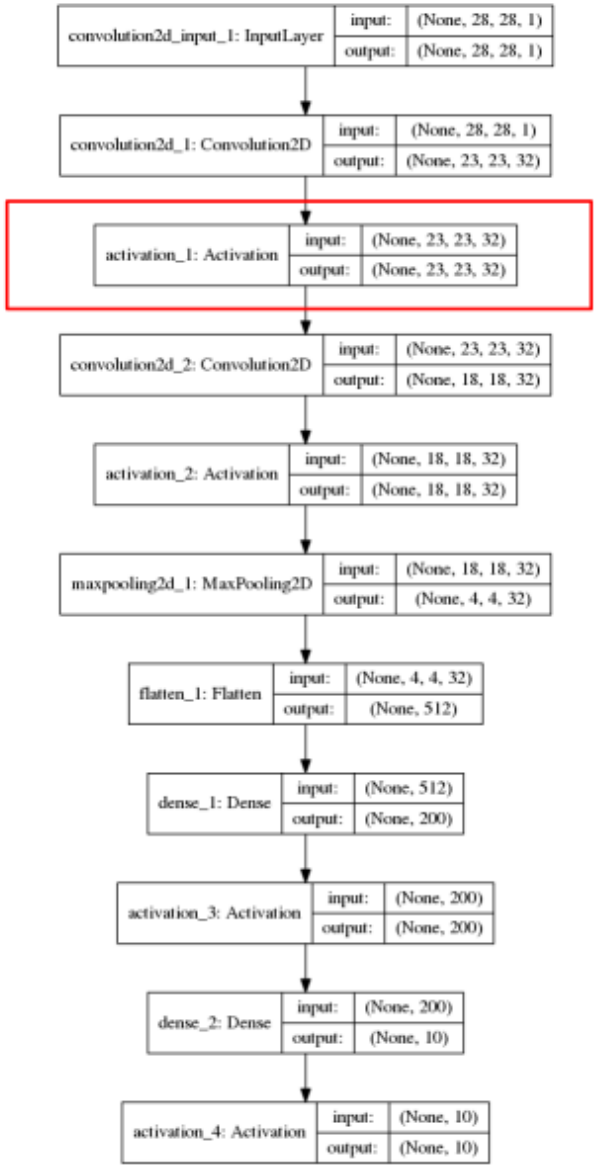
Resultado na base de teste

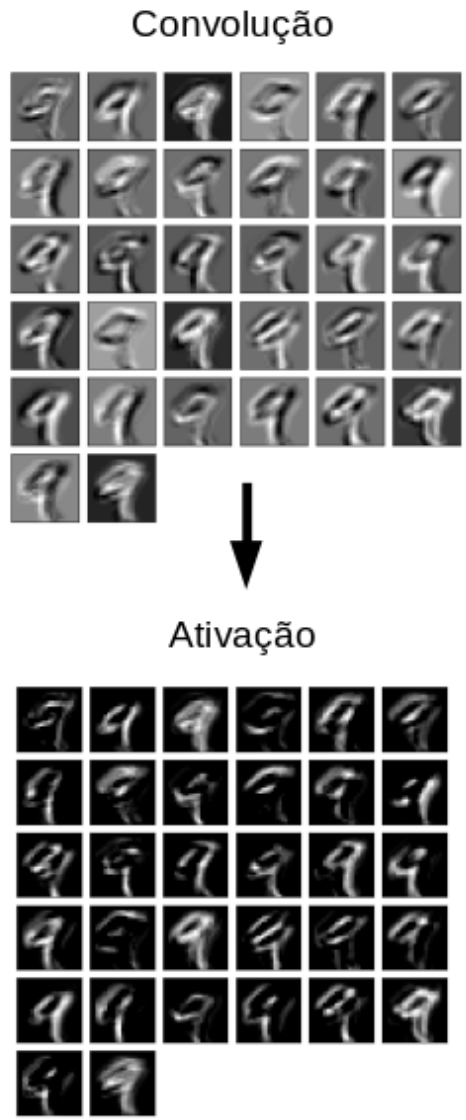
- 98.02%

Convolução - 1

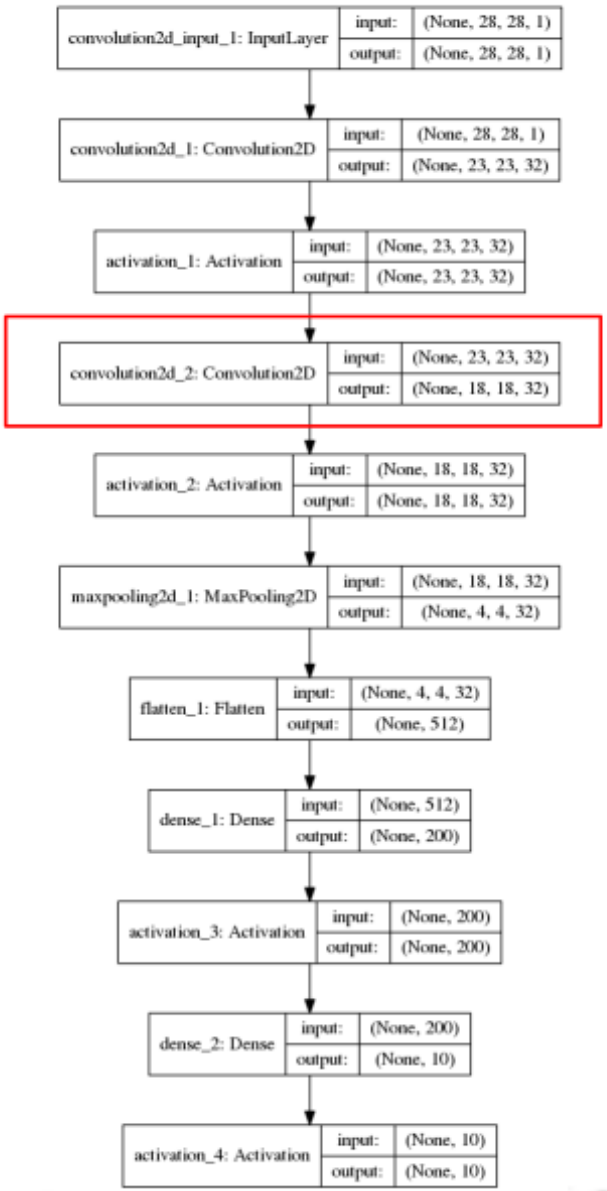


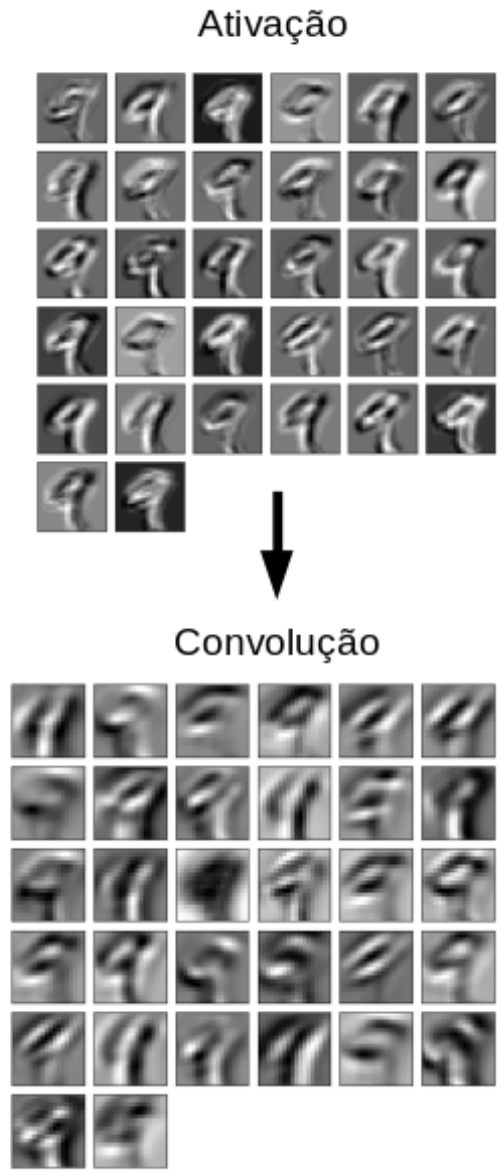
Ativação - 1

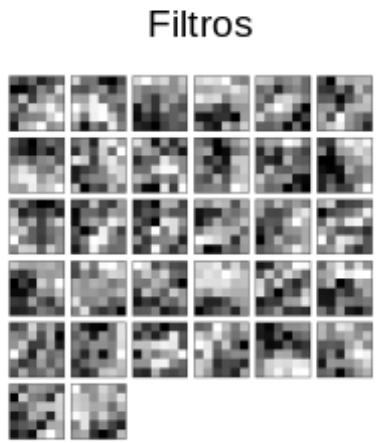

convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)


Convolução
Ativação

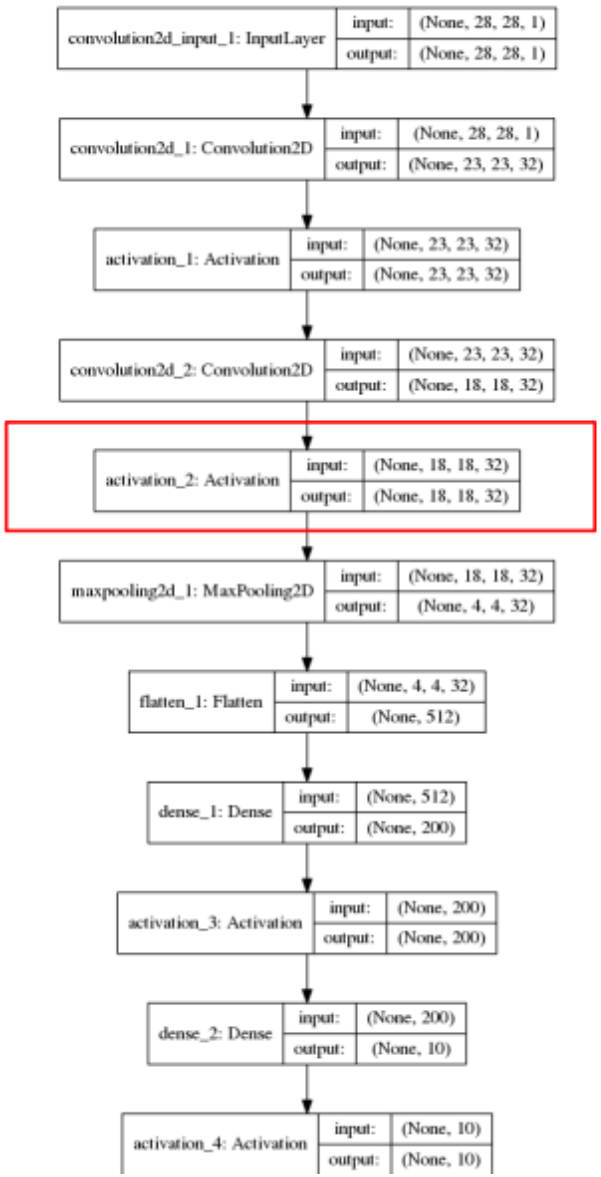
Convolução - 2

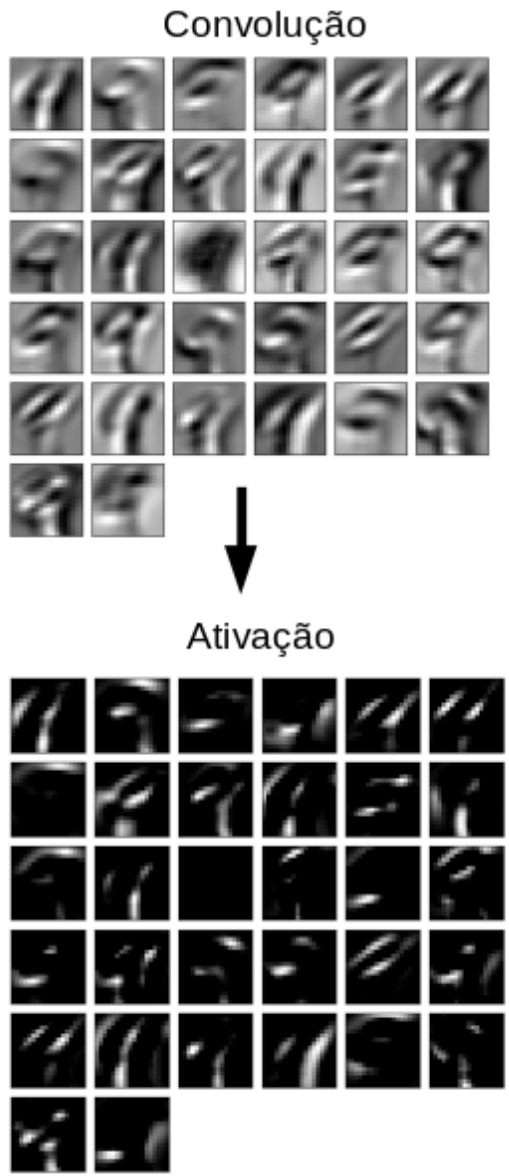




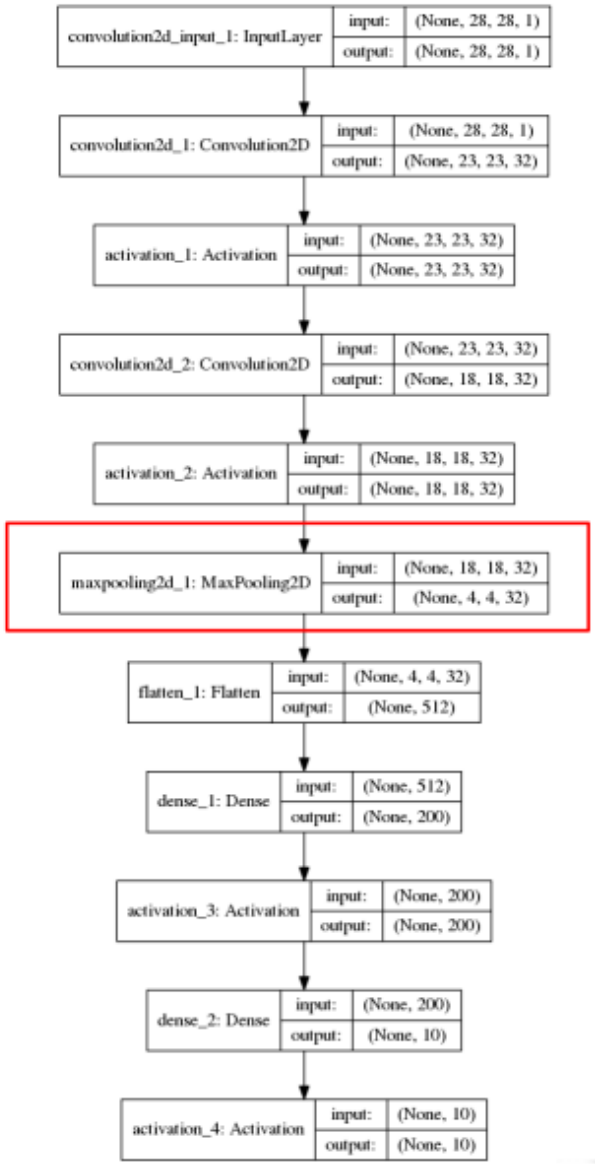


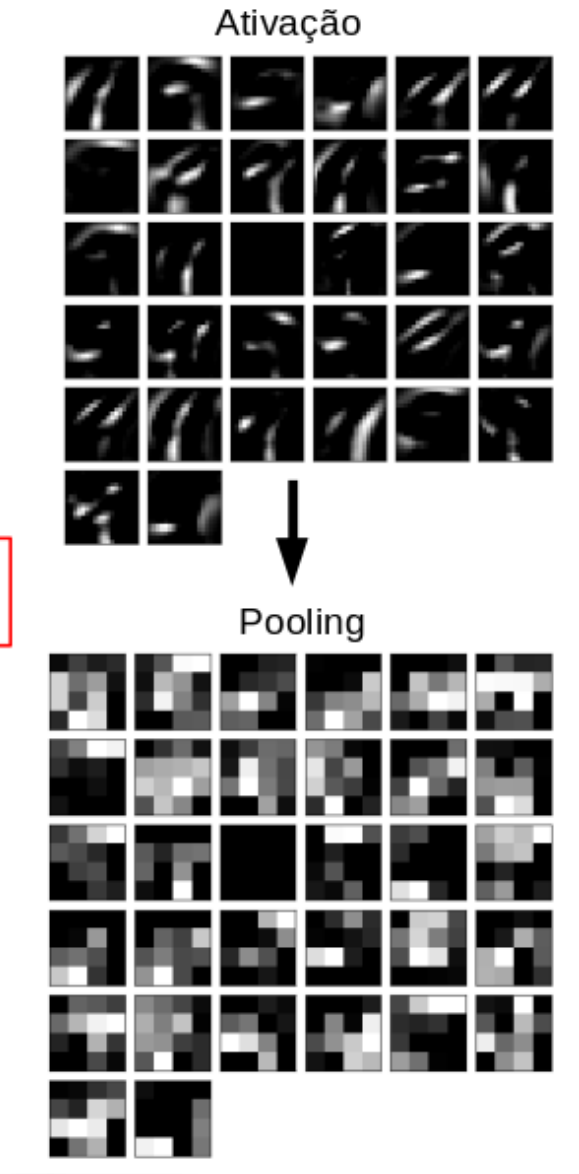
Ativação - 2


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)

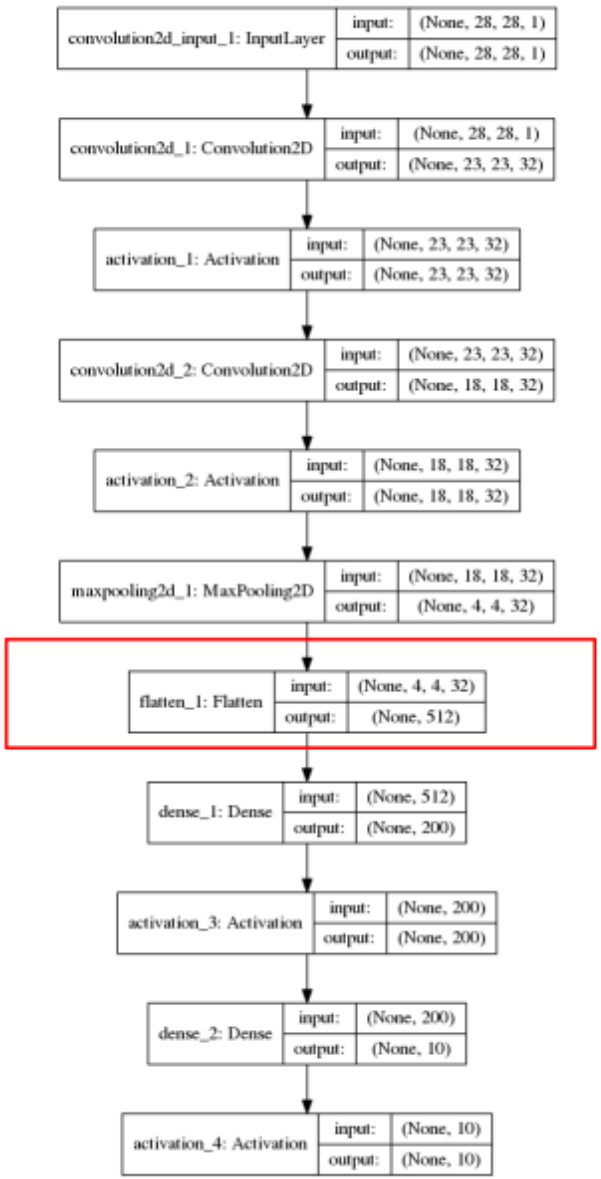

Convolução
Ativação

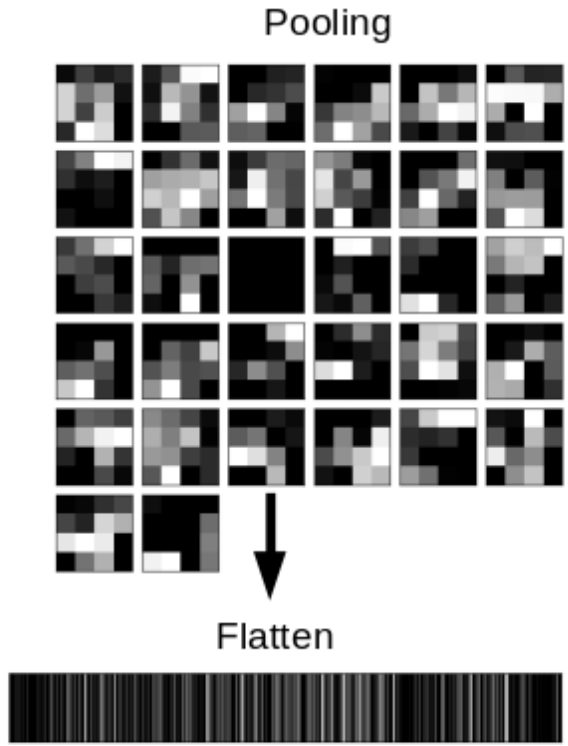
Pooling


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)

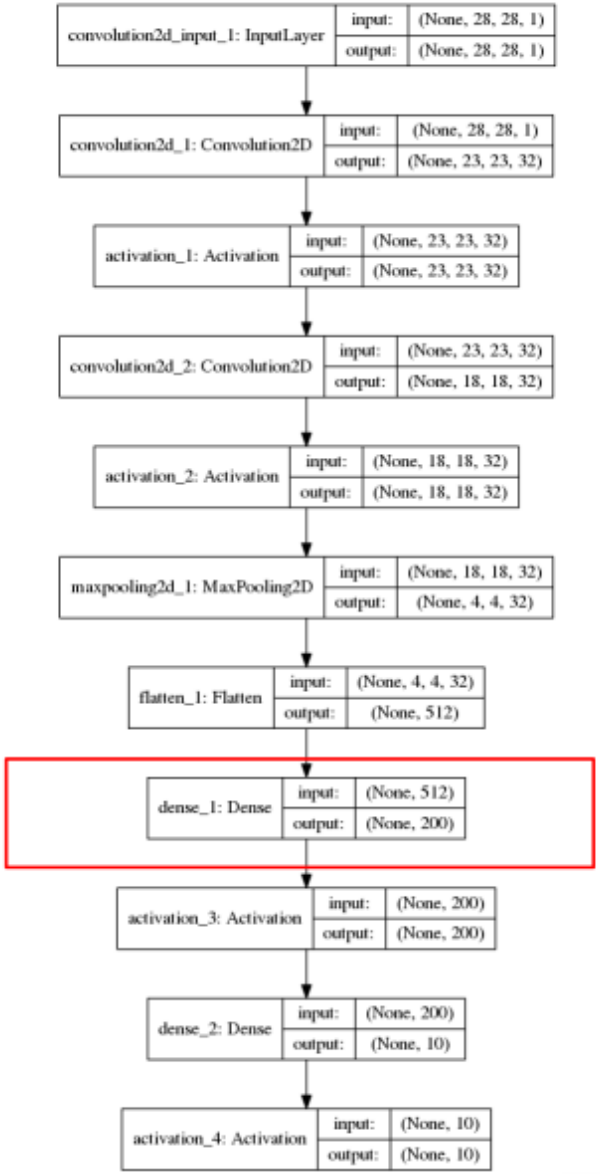

Ativação
Pooling

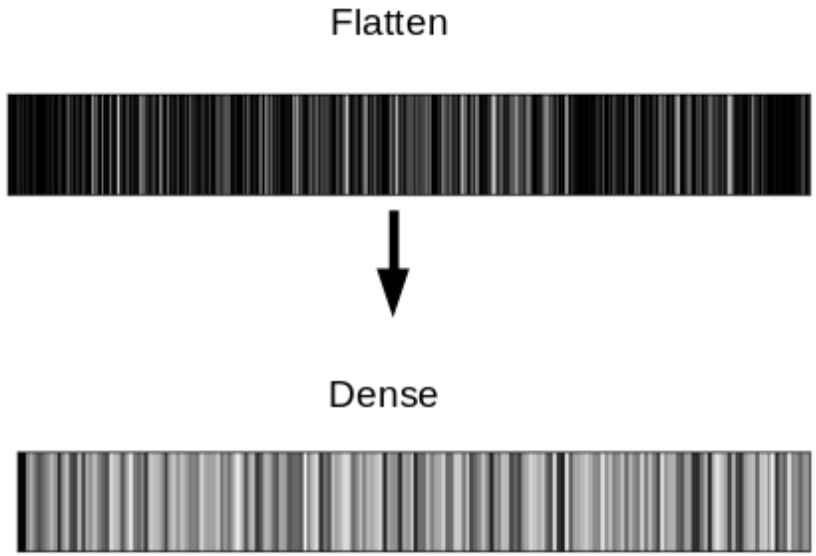
Flatten (N * 2D -> 1D)



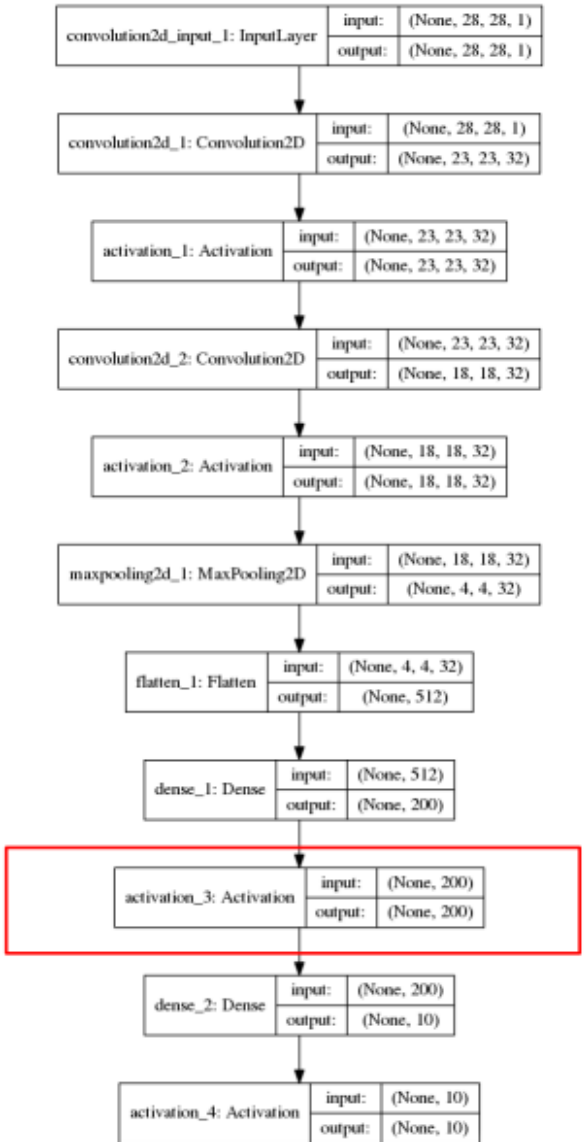


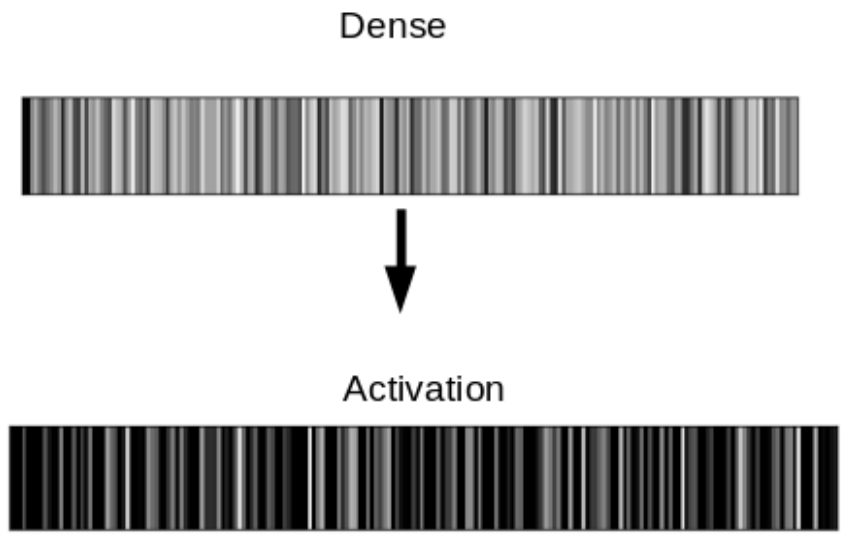
Dense - 1


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)

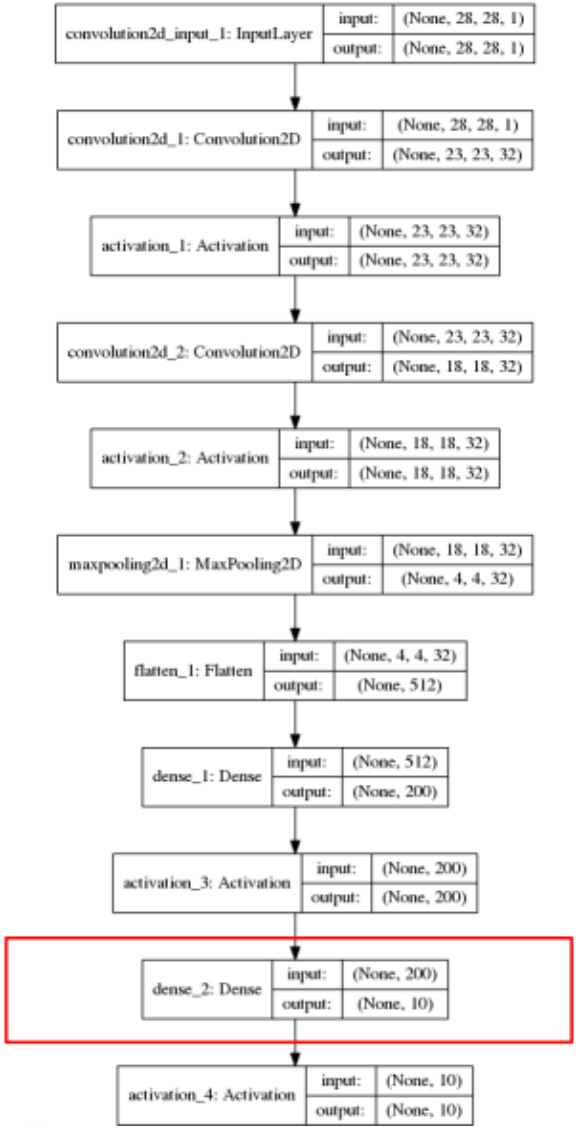

Flatten
Dense

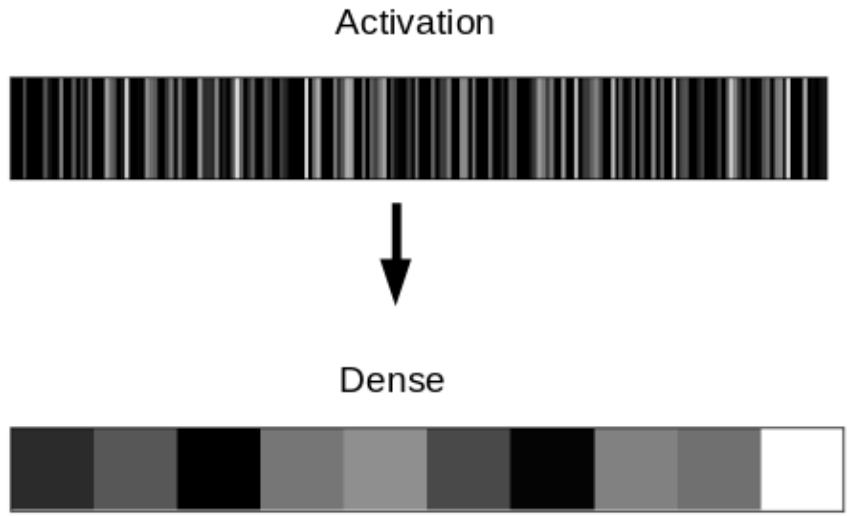
Ativação - 3


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)

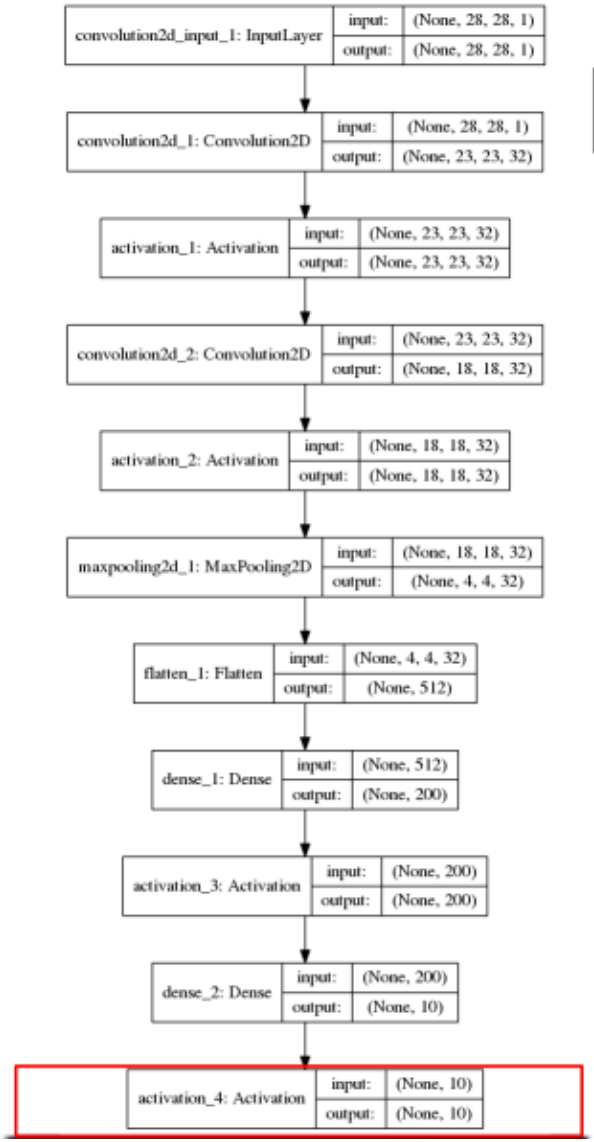

Dense
Activation

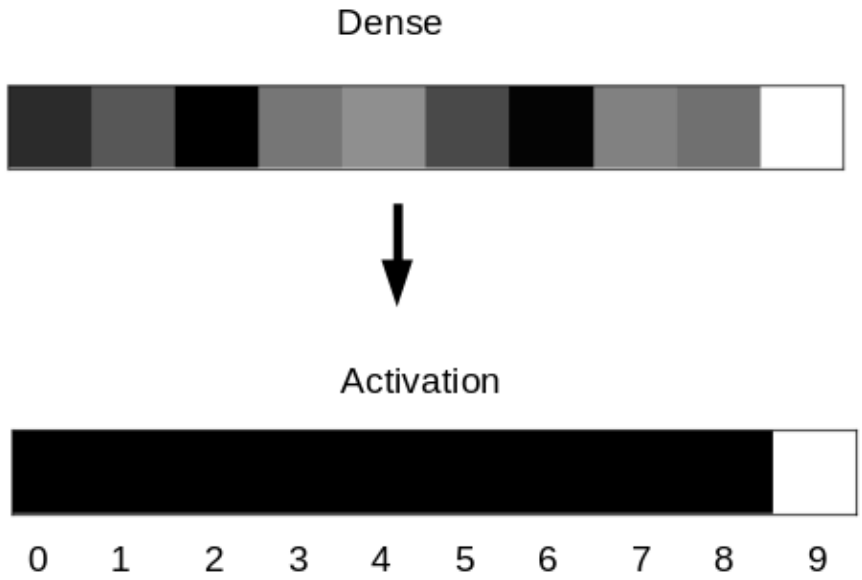
Dense - 2


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)


Activation
Dense

Ativação - 4


convolution2d_input_1: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
convolution2d_1: Convolution2D
input: (None, 28, 28, 1)
output: (None, 23, 23, 32)
activation_1: Activation
input: (None, 23, 23, 32)
output: (None, 23, 23, 32)
convolution2d_2: Convolution2D
input: (None, 23, 23, 32)
output: (None, 18, 18, 32)
activation_2: Activation
input: (None, 18, 18, 32)
output: (None, 18, 18, 32)
maxpooling2d_1: MaxPooling2D
input: (None, 18, 18, 32)
output: (None, 4, 4, 32)
flatten_1: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 200)
activation_3: Activation
input: (None, 200)
output: (None, 200)
dense_2: Dense
input: (None, 200)
output: (None, 10)
activation_4: Activation
input: (None, 10)
output: (None, 10)

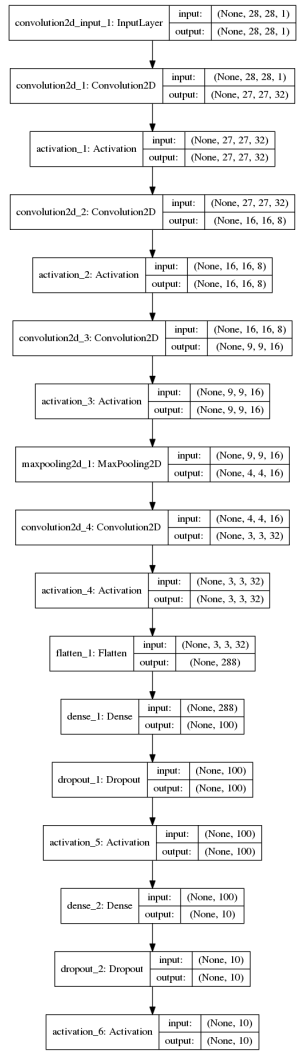

Dense
Activation
0 1 2 3 4 5 6 7 8 9

Resultados das demais redes testadas - 1

Precisão na base de treino: 98.94%

Precisão na base de teste: 98.89%

3 conv + 1 pooling + 3 conv + 1 pooling + 2 FC

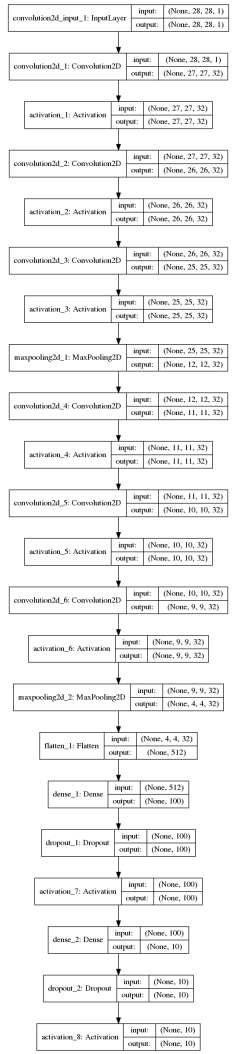


Resultados das demais redes testadas - 2

Precisão na base de treino: 98.94%

Precisão na base de teste: 99.06%

3 conv + 1 pooling + 3 conv + 1 pooling + 2 FC *comdropout*



MLP & CNN

- CNN é uma extensão do conceito da MLP
- Convoluções e Pooling ajudam a diminuir rapidamente o número de variáveis do sistema
- Próprio para o processamento de imagens e vídeos